LANGAGE PYTHON, RACINE ET MÉTHODE DE NEWTON

PETER PHILIPPE

Circonscrire l'immense œuvre scientifique de Sir Isaac Newton, ce célèbre savant anglais du 17e siècle, à une soi-disant pomme tombée sur sa tête [1] qui lui aurait fait germer l'idée de sa fameuse loi sur l'attraction universelle, est une offense cyclopéenne à ses travaux de recherche.

Newton c'est donc de la physique, en mécanique céleste notamment, mais également de la mathématique, qui ne se souvient pas de la formule de son fameux binôme enseigné au lycée $(a+b)^2 = a^2 + 2ab + b^2$? Il est aussi l'auteur d'une méthode d'interpolation faisant appel aux différences divisées. Ses contributions majeures sont nombreuses, l'une d'entre elles, appartient à l'analyse dite « infinitésimale [2] ». La merveilleuse méthode qui va nous intéresser tout particulièrement dans ses lignes, permet de rechercher, mais plus objectivement, d'approximer la racine réelle, notée x^* d'une fonction polynomiale $f(x)$.

La fonction candidate f se doit d'être *lisse* (cela sous-entend sans aucune valeur absolue), définie dans l'ensemble des nombres réels $\mathbb{R}$ et dérivable au moins deux fois (je me limiterai à préciser que la condition de classe C^2 est exigée, sans rentrer dans les détails). L'expression de cette méthode se construit simplement avec la formule de Taylor-Young, à laquelle il suffit de supprimer les termes supérieurs au premier ordre. La célèbre version itérative prend la forme suivante :

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \text{ avec } i = 0, 1, 2, \ldots$$

Cette méthode, dite « à un pas », a donc été imaginée par Sir Isaac Newton en 1669, la version itérative provient des efforts postérieurs de Joseph Rapshon en 1690. Dans les siècles qui ont suivi, cela donna naissance à une pléthore d'autres méthodes. De nos jours encore, la recherche est toujours active pour cette branche de l'analyse, le nombre de publications pour les méthodes basées sur ce principe est assez élevé. D'ailleurs, dans l'un de mes dossiers, il y a plus de 500 articles au format PDF rien que pour ce sujet et je suis certainement très loin du compte par rapport à ce qui est réellement publié.

Le principe sous-jacent de cette méthode est, en apparence seulement, d'une simplicité déconcertante, presque provocante, cela en laisse même pantois d'admiration. Exhibons

1. Cette légende est d'ailleurs fausse.

2. Comme tant d'autres de ses contemporains Leibniz, Taylor, Lagrange, Maclaurin, Rolle, Bernoulli ou encore Laplace qui ont apporté des innovations primordiales dans ce domaine fondamental des mathématiques.

son fonctionnement avec un petit exemple, et pour cela, prenons tout d'abord une fonction qui n'invite pas vraiment à la causerie, mais sans être pour autant haïssable :

$$f(x) = 2x^5 - x^4\cos(2x) + x^2\arctan(x+1) - 2\sqrt{x+1} - \sin(x) - 1 = 0$$

Le degré dominant de cette fonction étant impair, elle aura au minimum une racine réelle en comptant son ordre de multiplicité, les éventuelles racines supplémentaires pourront alors être réelles $\mathbb{R}$ ou complexes $\mathbb{C}$. Ici, pour cette misérable note, seules les racines réelles seront exposées.

Passons maintenant au calcul de sa dérivée au premier ordre $f'(x)$, cette dernière est plutôt longue du fait de la présence de plusieurs termes trigonométriques. Mais avec la connaissance de certaines règles de dérivations élémentaires, puis après avoir pris une bonne inspiration, cela se passe généralement bien :

$$f'(x) = 10x^4 - 4x^3\cos(2x) + 2x^4\sin(2x) + \frac{x^2}{(x+1)^2+1} - \frac{1}{\sqrt{x+1}} + 2x\arctan(x+1) - \cos(x)$$

Cette méthode étant itérative, on commence alors par définir une valeur de départ pour notre racine, que l'on suppose pas trop éloignée de la racine recherchée x^*, optons pour $x_0 = 2$, au risque que cela cause un certain préjudice et par conséquent, une divergence nous entraînant irrémédiablement quelque part dans $\mathbb{R}$. Voici maintenant le petit code source en langage Python de cette fonction, sans fioritures, ni librairie additionnelle, ainsi qu'un exemple avec notre séduisante fonction :

```
from sys import float_info
from math import fabs, sin, cos, atan, sqrt

def newton_rahpson(
        f: callable,
        fd: callable,
        x0: float,
        itmax: int = 20
    ) -> None:

    xold = x0
    xnew = xold
    it = 0
    tol = float_info.epsilon

    while it <= itmax:
        fx = f(xnew)
        dfx = fd(xnew)
        if dfx < tol:
            print("Risque de division par zéro, sortie de la fonction.")
            exit
        xnew = xold - fx / dfx
        delta = fabs(xnew - xold)
        if delta < tol:
            print(f"\n--> Racine trouvée, x = {xnew} à {delta} près.")
            break
        print(f"Itération numéro {it:3d} -> x* = {xnew}")
        xold = xnew
        it += 1

# Pas de return (d'où le -> None), car ici récupérer le résultat n'est pas utile.
```

```
def fonction(x: float) -> float:
    ret = 2*x**5 - x**4 * cos(2*x) + x**2 * atan(x + 1) - sin(x) \
        - 2 * sqrt(x + 1) - 1
    return ret

def derivee(x: float) -> float:
    ret = 10*x**4 + 2*x**4 * sin(2*x) - 4*x**3 * cos(2*x) \
        + x**2 / ((x + 1)**2 + 1) \
        - 1 / sqrt(x + 1) - cos(x) + 2*x * atan(x + 1)
    return ret

if __name__ == "__main__":
    x0 = 2
    newton_rahpson(fonction, derivee, x0)
```

L'exécution de ce court programme affiche instantanément l'approximation recherchée :

```
Itération numéro   0 -> x* = 1.5425226962349505
Itération numéro   1 -> x* = 1.2664928053813442
Itération numéro   2 -> x* = 1.1196418131810073
Itération numéro   3 -> x* = 1.0725169943246782
Itération numéro   4 -> x* = 1.0679675865377685
Itération numéro   5 -> x* = 1.067927692170306
Itération numéro   6 -> x* = 1.067927689124829

--> Racine trouvée, x = 1.067927689124829 à 0.0 près.
```

Après l'exécution, on s'émerveille à la vue des itérations, en voyant la convergence se réaliser rapidement, nous amenant ainsi à l'approximation de la racine tant convoitée $x^* \approx 1,067927689124829$, car $f(x^*) \approx 0$ avec la tolérance offerte par le format `float` de Python, c'est-à-dire $2,220446049250313e-16$. La valeur initiale x_0 était pourtant assez éloignée de la racine recherchée et il n'était donc nullement assuré que la méthode puisse converger. Sans omettre, qu'il est impossible de trouver la racine exacte pour cette nature de fonction.

Pourtant, un tel scénario c'est lorsque tout se passe bien, les yeux écarquillés devant un spectacle féérique après avoir englouti un gros dénivelé... ou une bonne mousse au chocolat. En pratique, ce n'est pas systématiquement le cas et ce, pour plusieurs raisons comme une valeur initiale trop éloignée du fait de n'avoir aucun visuel sur la fonction, une racine multiple et non simple, des racines trop rapprochées, la présence de racines complexes, la nature histérique de la fonction générant des variations brutales de la tangente lors du passage à un point d'inflexion, ou encore un problème de précision informatique insuffisante.

L'étude de la convergence de cette méthode a fait couler beaucoup d'encre et cela continuera encore. Le fameux théorème de Kantorovich sur la convergence semi-locale publié en 1948, fit l'effet d'une véritable bombe, amenant à repenser totalement les coulisses ténébreuses de cette méthode qui, d'un seul coup, devenait nettement plus compliquées, laissant entrevoir, un tout nouvel univers à comprendre qui occupera encore très longtemps des cerveaux inapaisables.